한국산업정보학회논문지 제4권 제4호 (99.12)

Efficient Parallel Algorithm for Gram-Schmidt Method

Sun-Kyung Kim*

요 약 선형독립인 소수의 벡터들을 직교화 시키는 방법인 Gram-Schmidt 알고리즘, 몇 개의 극단적인 고유치를 구하는 방법인 란초스 알고리즘방법 등이, 한번의 반복동안 한번의 동기점만 가지도록 재구성된다. 즉 메시지 패싱 분산시스템에서 모든 프로세서들 사이에 한번의 통신만 요구되도록 알고리즘을 변화시킨다. 이러한 알고리즘들은 재구성되었다고 하며, 전통적인 방법에 비하여 더 나은 병렬성질을 가지게 된다.

Abstract Several Iterative methods are considered, Gram-Schmidt algorithm for thin orthogonalization and Lanczos methodfor a few extreme eigenvalues. For these methods, a variants of method is derived for which only one synchronization point per one iteration is required; that is, one global communication in a message passing distributed-memory machine per one iteration is required. The variant is called restructured method, and restructured method has better parallel properties to the conventional method.

1. Introduction

The Gram-Schmidt(GS) method can be used to compute the thin QR factorization V=QR directly. Two orthogonalizations are considered: Classical and modified methods.[1,2] Modified method is more stable compare to classical method. These methods have two basic types of operations: inner products and the vector updates. The inner products require global communication on a message passing system, so these dominate the computation. However the inner products in the conventional GS method cannot be performed in parallel, because they are computed in two different places every iteration.

Many important scientific and engineering problems require the computation of a small number of eigenvalues of large sparse symmetric matrices. The most commonly used algorithm based upon projection process to the Krylov subspaces for solving such an eigenproblem is the Lanczos algorithm.[3] In this method the inner products cannot be performed in parallel either, because of the same reason as the GS method.

* Department of Computer Engineering, Taegu University

In order to use parallel computers in specific applications, algorithms need to be developed and mapped on to a parallel computer architecture.[4,5,6] In this paper we restructure the conventional above methods to improve parallel performance in the orthogonalization of linearly independent vectors v1, v2 .... vm. The outline of this paper is as follows. In section 2, we review the basic properties of the classical and modified GS methods and Lanczos method. In section 3, we restructure the conventional GS method and Lanczos algorithm for better performance on parallel computers. In section 4, Comparisons among the conventional, restructured methods are discussed.

2. The Gram-Schmidt method

2.1 The classical GS method

We now discuss two alternative methods that can be used to compute the thin QR factorization V=QR, $V \in \mathbb{R}^{n \times m}$, directly. If rank(V)=m, then equation $v_k = \sum_{i=1}^{k} r_{ik} q_i \in span\{q_1, q_2, \ldots q_k\})$ can be solved for q_k :

$$q_k = (v_k - \sum_{i=1}^{k-1} r_{ik} q_i) \ / \ r_{kk}$$

Thus, we can think of qk as a unit 2-norm vector in the direction of

$$z_k = v_k - \sum_{i=1}^{k-1} r_{ik} q_i$$

where to ensure $z_k \in span\{q_1, \cdots, q_{k-1}\}^{\perp}$
we choose

$$r_{ik} = q_i^T a_k \qquad i = 1,\ldots,k$$

This leads to the classical Gram-Schmidt(CGS) algorithm for computing V=QR.[1] The classical Gram-Schmidt method is as follows (Here, we use the same array for V and Q)

Algorithm 1. The CGS method

```
1. compute (q1, q1)
2. $r_{11} = (q_1, q_1)^{\frac{1}{2}}$
3. q1 = q1 / r11
4. For j = 2 to m
   4.1 For k=1 to j-1
         compute (qk, qj) :inner product
           $r_{kj} = (q_k, q_j)^{\frac{1}{2}}$
       EndFor
   4.2 For k=1 to j-1
         qj = qj - rkj · qk
       EndFor
   4.3 compute (qj, qj) :inner product
         $r_{jj} = (q_j, q_j)^{\frac{1}{2}}$
         qj = qj / rjj
   EndFor
```

In the j-th step of CGS, the jth columns of both Q and R are generated.

2. 2 The modified GS method

Unfortunately, the CGS method has very poor numerical properties in that there is typically a severe loss of orthogonality among the computed qk. Interestingly, a rearrangement of the calculation, known as modified Gram-Schmidt(MGS), yields a much sounder computational procedure. In the k-th step of MGS, the k-th column of Q(denoted by qk) and the k-th row of R(denoted by r_k^T) are determined where $Q \in IR^{n \times m}$ has orthonormal columns and $R_1 \in R^{n \times m}$ is upper triangular.[1,2]

Algorithm 2. The MGS method

```
1. compute (q1, q1)
2. $r_{11} = (q_1, q_1)^{\frac{1}{2}}$
3. q1 = q1 / r11
4. For j=2 to m
   4.1 For k=j to m
         compute(qj-1, qk) :inner product
         $r_{j-1,k} = (q_{j-1}, q_k)^{\frac{1}{2}}$
       EndFor
   4.2 For k=j to m
         qk = qk - qj-1 · rj-1,k
       EndFor
   4.3 compute(qj, qj) :inner product
         $r_{jj} = (q_j, q_j)^{\frac{1}{2}}$
         qj = qj / rjj
   EndFor
```

2. 3 The Lanczos method

Many important scientific and engineering problems require the computation of a small number of eigenvalues of large sparse symmetric matrices. The most commonly used algorithms based upon projection process to the Krylov subspaces for solving such an eigenproblem are the Lanczos algorithm. The basic Lanczos procedure can be viewed as the Gram-Schmidt orthogonalization of the Krylov subspace basis $\{q_1, Aq_1, \ldots, A^{j-1}q_1\}$. Furthermore, for each j, $T_j = Q_j^T A Q_j$ is the orthogonal projection of A onto the subspace spanned by the Lanczos vectors $Q_j = q_1, \ldots, q_j$ such that $Q_j^T Q_j = I_j$ where I_j is the identity matrix of order

j. The eigenvalues of the Lanczos matrices T_j are called Ritz values of A in Q sub j. For many matrices and for relatively small j, several of the extreme eigenvalues of A, that is several of the algebraically-largest or algebraically-smallest of the eigenvalues of A, are well approximated by eigenvalues of the corresponding Lanczos matrices.[3] The standard Lanczos algorithm is as follows:

Algorithm 3. The Lanczos method

```
Choose $q_1$ with $||q_1||_2 = 1$, $q_0 = 0$
For j=1 until Convergence Do
  1. compute and store $Aq_j$
  2. compute $(Aq_j, q_j)$ :inner product
     $\alpha_j = (Aq_j, q_j)$
  3. $r_j = Aq_j - \beta_{j-1}q_{j-1} - \alpha_j q_j$
  4. compute $(r_j, r_j)$ :inner product
     $\beta_j = \sqrt{(r_j, r_j)}$
  5. $q_{j+1} = r_j / \beta_j$
EndFor
```

3. The restructured method

3.1 Restructuring GS method

The classical GS algorithm has two basic types of operations: inner products and the vector updates. In algorithm 1,2 the inner products cannot be performed in parallel. The algorithms based on restructuring the conventional GS algorithms are introduced here. The restructured algorithm decreases the global communication costs and thus get the better performance in a distributed-memory message passing systems. For shared memory systems with a few processors, processor synchronization is fast but accessing data from the main memory may be slow. Thus, the data localities of the basic operation parts of the GS algorithm determine the actual execution time of the algorithm. In algorithm 1, step 4.1 must be completed before the rest of the computations in the same step start. Also in algorithm 2, Synchronization points are two, that is, step 4.1 and step 4.3. This forces double accesses of vector q from the main memory at each iteration. Algorithm 4,5 are variants of algorithm 1,2 for which are derived for which only one synchronization point per one iteration is required. In the restructured method, the inner products needed for one iteration can be performed simultaneously. so the algorithm 4,5 are more suitable for parallel processing.

Algorithm 4. The restructured CGS algorithm

```
1. For j=1 to m-1
   1.1 compute(qj, qj)
   1.2 For k=1 to j  :all inner products
         compute (qk, qj+1)
       EndFor
   1.3 $r_{jj} = (q_j, q_j)^{\frac{1}{2}}$
       rj,j+1 = (qj, qj+1) / rjj
   1.4 For k=1 to j
         rk,j+1 = (qk, qj+1)
       EndFor
   1.6 qj = qj / rjj
   1.7 For k=1 to j
         qj+1 = qj+1 - qk · rk,j+1
       EndFor
   EndFor
2. compute(qm, qm)
   $r_{mm} = (q_m, q_m)^{\frac{1}{2}}$
   qm = qm / rmm
```

Algorithm 5. The restructured MGS algorithm

```
1. For j=1 to m-1
   1.1 For k=j to m :all inner products
         compute(qj, qk)
       EndFor
   1.2. $r_{jj} = (q_j, q_j)^{\frac{1}{2}}$
   1.3. For k=j+1 to m
          compute(qj, qk)
          rjk = (qj, qk)/rjj
        EndFor
   1.4 qj = qj / rjj
```

```
    1.5 For k=j+1 to m
          qk = qk - qj · rjk
        EndFor
    EndFor
2. compute(qm, qm)
    $r_{mm} = (q_m, q_m)^{\frac{1}{2}}$
    qm = qm / rmm
```

The orthogonal vectors q_j in Algorithm 4,5 are generated in the same way as the conventional GS method. Computationally the difference between Algorithm 1,2 and 4,5 is in the computation of R. We need more extra scalar operations to compute r_{jj}, r_{jk} in the algorithm 4,5. However, Algorithm 4,5 seems more promising than Algorithm 1,2 for parallel processing because the inner products required to advance each iteration can be executed simultaneously. Also, one memory sweep through the data is required to complete each iteration allowing better management of slower memories in a memory hierarchy computer.

3.2 The restructured Lanczos method

In the algorithms 3, the inner products cannot be performed in parallel. Algorithms based on restructuring the standard Lanczos algorithms to decrease the global communication cost and to get better performance in distributed-memory message passing systems, is introduced here. For shared memory systems with few processors, processor synchronization is fast but accessing data from the main memory may be slow. Thus the data localities of the three basic operation parts of the Lanczos algorithm determine the actual time complexity of the algorithm. The data locality of the restructured Lanczos algorithms is better than that of the algorithm 3. In algorithm 3, step 2 and step 5 are two synchronization points. This forces double access of vectors q, Aq, r from the main memory at each symmetric Lanczos iteration.

Algorithm 6. The restructured Lanczos Algorithm

```
Choose w $r_0$ith $r_0 \neq 0$, $q_0 = 0$
For j=0 until Convergence Do
  1. compute and store $Ar_j$
  2.        compute        $(r_j, r_j), (Ar_j, r_j)$
            : all inner products
     $\beta_j = \sqrt{(r_j, r_j)}$
     $\alpha_{j+1} = (Ar_j, r_j)/(r_j, r_j)$
  3. $q_{j+1} = r_j/\beta_j$
  4. $r_{j+1} = Ar_j/\beta_j - \beta_j q_j - \alpha_{j+1} q_{j+1}$
EndFor
```

Algorithm 6 is a variant of algorithm 3 and the orthonormal vectors q_j are generated in the same way as the standard Lanczos method. Computationally the difference between Algorithm 3 and 6 is the computation of α_j, r_j. The computation of β_j is the same in algorithms 3 and 6. Loss of orthogonality from a very small residual vector is unavoidable in any of the algorithms. We need one more vector operation to compute r_j in Algorithm 6. Algorithm 6 seems more promising for parallel processing because the two inner products required to advance each iteration can be executed simultaneously. Also, one memory sweep through the data is required to complete each iteration allowing better management of slower memories.

If reorthogonalization is used in algorithms 3 and 6, another synchronization point is required in each algorithm. But it does not affect the fact that two inner products to compute α_j, β_j can be executed simultaneously in algorithm 6.

4. Comparisons of methods

In the algorithms 1,2,4,5 the inner products are m(m+1)/2, the vector updates are m(2m-1)/2, and the storage requirements are m, that is, these algorithms require same number of the vector operations and same amount of storage. In the GS

<Table 1> Comparison of conventional method & restructured method.

	CGS	Restructred CGS	MGS	Restructured MGS
Synchronization points	2m-1	m	2m-1	m
Scalar products	$\frac{m(m+1)}{2}$	$\frac{(m+2)(m+1)}{2}$	$\frac{m(m+1)}{2}$	m^2

<Table 2>. Time performance(msec) per dot product.

P	4	8	16	32	64
Conventional	27.454	15.929	11.050	9.729	11.020
Restructured	19.331	10.5	7.421	6.601	7.908

<Table 3>. Dot product rate in Lanczos algorithm.

P	4	8	16	32	64
Conventional	28%	30%	37%	48%	64%
Restructured	23%	24%	30%	41%	59%

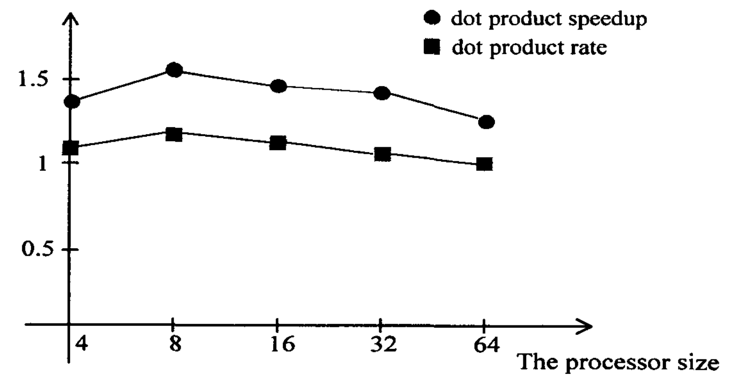

Figure 1. dot product speedup for standard/restructured

method(algorithms 1,2) $m(m+1)/2$ inner products should be separately repeated 2m-1 times and so communication time rate is high for these QR factorization processes. Algorithms 4,5 are variant of algorithms 1,2, algorithm 4,5 are derived for which only one synchronization point per one iteration is required. In the restructured method, the inner products needed for one iteration can be performed simultaneously. so the algorithm 4,5 is more suitable for parallel processing compare to algorithm 1,2.

Table 1 shows the global communications during m iterations of the corresponding conventional and restructured GS method. During the procedure of the conventional GS algorithm in the section 2, inner products per one iteration should be separately computed. However, in the restructured algorithm all inner products needed for one iteration can be performed simultaneously. So the computation time and communication time necessary for inner products are reduced by a factor of 1/2 roughly compared to the conventional method.

On message passing system like Hypercube, IBM SP, lots of times are needed for global communication, which in turn reduces the efficiency of the parallel system.
Table 2 shows the average time performance per one inner product in case that two inner products are performed seperately or simultaneously on hypercube machine. Table 3 shows the rate of inner product in the algorithm 3,6. So simultaneous computation of inner products is much better parallel properties. Figure 1 shows the speedup performance of dot product in the restructured algorithm.

5. Conclusions

As the number of the processors involved in the parallel system is increased, the relative importance of the communication cost grows. In this paper, we proposed the restructured Gram-Schmidt method, and the new method is suitable to reduce the communication cost. The restructured method orthonormalizes the vectors in the same way as the conventional method, but the new one is more effective in the parallel system because a large amount of the inner products can be done at once. This process can reduce the data communication time in a message passing system. The parallel algorithm also helps to reduce the memory latency time in shared memory systems by reducing the synchronization point showing a better data locality.

References

[1] G. H. Golub, C. F. Van Loan, MATRIX Computations, Johns Hopkins University Press. (1996)

[2] James W. Demmel, Applied Numerical Linear Algebra, the Society for Industrial and Applied Mathematics press. (1997)

[3] Jane K. Cullum and Raph A. Willoughby, Lanczos Algorithms for Large Symmetric Eigenvalues Computation, Birkhauser Boston, Inc. (1985).

[4] Sun Kyung Kim and A. T. Chronopoulos, "A Class of Lanczos Algorithms Implemented on Parallel Computers", Parallel Computing 17 (1991) pp 763-778.

[5] Pontus Matstoms, "Parallel sparse QR factorization on shared memory architectures", Parallel Computing 21 (1995) 473-486

[6] Paul E. Saylor, "Leapfrog Variants of Iterative Methods for Linear Algebraic Equations", Journal of Computational and Applied Mathematics 24 (1988) pp 169-193.

김 선 경

1979. 2 이화여자대학교 수학과 졸업
1982. 2 한국과학기술원 전산학과 졸업(석사)
1991. 7 University of Minnesota, Computer Science 박사

인하대학교 전산학과 조교 및 강사(1982.3-1984.2)
계명대학교 전산학과 강사(1984.3-1984.8)
University of Minnesota, Computer Science 조교 (1988.3-1991.7)
대구대학교 컴퓨터정보공학부 부교수(1992.3-현재)